DETAILED ACTION

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on October 16, 2009 has been entered.

2. Applicant's amendment dated October 16, 2009, responding to the Final Office Action mailed on February 19, 2009 and the Advisory Action dated on August 19, 2009 respectively provided in the rejection of claims 1, 3-7, 15, 17-22, and 24-28, wherein claims 1, 15, and 22 have been amended; and claims 29-34 have been newly added.

Claims 1, 3-7, 15, 17-22, and 24-34 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are not persuasive. Please see the section of "Response to Arguments" for details.

Response to Arguments

3. Applicant's arguments filed on October 16, 2009 have been fully considered but they are not persuasive.

In the remarks, Applicant argues that, for examples:

(**A.1**) Applicant contends that the Advisory Action fails to address the Applicant's previously submitted assertions that at least the elements A and B (brief on page 12, second paragraph through page 13) detailed above are: 1) included in the currently pending claims, 2) rejected under Gauthier or Liu, and 3) disclosed in the priority patent. As such, it <u>does not matter</u> that <u>other elements</u> of the pending claims may or <u>may not be disclosed in the priority patent</u> (brief on page 14, first paragraph – emphasis added)

Examiner Responses:

Examiner respectfully disagrees. Firstly, as pointed out by Applicant, the priority claim made in the present application properly established the present application is a continuation-in-part drawing priority to the '947 patent (brief on page 11, third full paragraph; also see the Advisory Action)

Secondly, as also indicated by Applicant, the Examiner, in the Advisory Action, newly introduces the argument that several specific limitations of the currently pending claims are <u>not fully disclosed in the '947 patent</u> and therefore <u>not</u> entitled to the priority date of the '947 patent (brief on page 11, third full paragraph)

Thirdly, in order to get priority date from a <u>continuation-in-part</u> parent application (i.e., the '947 patent), <u>each individual claim</u> <u>as a whole</u> in the pending claims (i.e., claims 1, 15, 22, and 29) is analyzed against supported parent

application(s). As indicated by Applicant, some other elements of the pending claims may not be disclosed in the priority patent (brief on page 14, first paragraph). For example, at least, one of the elements of '*task context block (TCB)*, '*a set of n tasks scheduled*' and '*a scheduling algorithm*' recited in each independent claim is <u>not fully supported in the '947 patent</u>.

Lastly, hence the priority date can not be applied to the pending claims. Therefore, <u>all the prior arts</u> applied in the previous Office Action still applicable to the current Office Action (emphasis added)

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made

4. Claims 1, 3, 5, 7, 15, 17, 19, 21-22, 24, 26, 28-30, 32, and 34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman et al. (US Patent 4,796,179) (hereinafter 'Lehman') in view of Gauthier et al. *'Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software'*, 2001, IEEE (hereinafter 'Gauthier'), Liu et al. *'Timed Multitasking for Real-Time Embedded Software'*, February 2003, IEEE (hereinafter 'Liu'), and Kuljeet Singh *'Design and Evaluation of an Embedded*

Real-time Micro-kernel', October 2002, pp. 1-133, Virginia Polytechnic Institute and State University (hereinafter 'Singh')

5. **As to claim 1** (Currently Amended), Lehman discloses a method for developing a real-time operating system (e.g., Fig. 1; Col. 1, Lines 46-48; Col. 4, Lines 63-68; Col. 5, Lines 1-2), comprising:

specifying a set of *n* tasks (e.g., Col. 1, Lines 33-38), *task(1)* through *task(n),* to be scheduled for execution; (e.g., Abstract, Lines 8-14; Col. 3, Lines 1-8; Col. 5, Lines 5-12; Col. 135, Lines 17-24).

Further, Lehman discloses specifying a scheduling algorithm (e.g., Col.3, Lines 36-39; Col. 9, Lines 56-61; Col. 32, Lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, Lines 29-32; Col. 16, Lines 21-23; Col. 2, Lines 36-39; Col. 9, Lines 62-68; Col. 10, Lines 1-2; Col. 32, Lines 5-54) for scheduling the execution of the set of *n* tasks.

Furthermore, Lehman discloses at least one of the task of the set of *n* tasks being selected as a preemptive or a non-preemptive task (e.g., Col. 9, Lines 52-56; Col. 10, Lines 3-12; Col. 35, Lines 1-5; Col. 136, Lines 40-50); synthesizing source code to implement a task scheduler (e.g., Fig. 4, element 24; Fig. 24; Fig. 26; Col. 2, Lines 36-39; Col. 3, Lines 36-39; Col. 9, Lines 56-61; Abstract, Lines 20-25; Col. 10, Lines 8-12) that uses the scheduling algorithm for controlling execution of said *n* tasks, but Lehman does not explicitly disclose other limitations stated below.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier discloses:

using a data processor to synthesize source code (e.g., Sec. 4.2 – Synthesis of Application Specific OS and SW targeting) from commands embedded in source code (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values) to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks (e.g., Fig. 1 – an example of OS-based SW implementation of multiple tasks – "scheduling"; Sec. 2 – Related Work, 2nd Para – there are three approaches in SW implementation from multi-task descriptions; the first two approaches user OS as a scheduler and an interface of multiple tasks to the target architecture; bullets 1 through 3); and

said synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of "Operating System Library", " Code Selector", "Code Expander", "Targeted Operating Systems Code"; Sec. 3.7 –

Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into the Lehman's system to further provide other limitations stated above in the Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3).

Furthermore, Lehman and Gauthier do not explicitly disclose other limitations stated below.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses:

specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as <u>an independent thread</u>, w<u>ith an init() method that initializes hardware resources</u> and a start() method that registered itself to the run-time system …; 4th Para - <u>Tasks</u> … <u>to init</u>() and start() …

stopExec() … A TASK data structure is also generated, which defines a scheduling entry representing a task …; Fig. 10 – Definition of a TASK in generated C code); the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., P. 71, 'Scheduling Analysis' - … be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman- Gauthier's system to further provide other limitations stated above in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

Lastly, Liu discloses the challenges of developing embedded software for real-time control systems and argued that timing properties should be introduced at the programming level to bridge the gap between algorithm development and real-time priority tuning, but Lehman, Gauthier, and Liu do not further explicitly disclose other limitations stated below.

However, in an analogous art of *Design and Evaluation of an Embedded Real-time Micro-Kernel*, Singh discloses:

each task having an associated task context block (e.g., Sec. 2.2.7.1.1 The Model, 1st Para - … associate with each task a context; an identification string or number; a status; and a priority if applicable. These items are stored in a structure called a task-control block (or TCB) ...; Sec. 2.2.7.1.2 Task States, 1st Para – The operating system manages the task-control blocks by keeping track of the status or state of each task …); and

synthesizing source code from commands embedded in source code to manipulate task context blocks for said set of t init-tasks (e.g., Fig. 6.2 – Process data structure; Sec. 6.1 ECO Implementation, 3rd Para - … The information about the handles for incoming and outgoing data channels is obtained through the parameter p, which is a part of the Task Control Block (TCB) for the thread that executes the ECO …; P. 116 - This module allocates and initializes the data channels and Process control Blocks during the OS startup phase …; P. 119 – *allocate_ initialize_ process* – Function which allocates space to the Process Control Blocks and initializes them …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Singh into the Lehman-Gauthier-Liu's system to further provide other limitations stated above in the Lehman-Gauthier-Liu system.

The motivation is that it would further enhance the Lehman-Gauthier-Liu's system by taking, advancing and/or incorporating the Singh's system which

offers significant advantages that it can take advantage of dual-register-set hardware to drastically reduce context-switching overhead; it also provides support for dynamic priorities, "firing rules" for specifying the data channel conditions; and the reconfigurable options provided by DARK facilitate selective removal of unneeded features to improve performance, without requiring any change to the applications as once suggested by Singh (e.g., Sec. 1.2 Problem Statement, 3rd Para)

6. **As to claim 3** (Previously Presented) (incorporating the rejection in claim 1), Lehman discloses the method and the apparatus further including specifying *f* *f-loop* tasks, each having an associated integer value *li* for *i* ranging from *1* to *f* and *f* being less than or equal to *n* (e.g., Col. 20, line 63 through Col. 21, line 20 – for loops using an incrementing or decrementing counter, i.e. Loop for *I* = 1 to *X* (executing) block of statements), said task scheduler addresses the task scheduler executing the loops including a continuously executing loop such that each *f-loop* task executes exactly once every *li* times that the loop is executed (e.g., Col. 21, Lines 13-19)

7. **As to claim 5** (Previously Presented) (incorporating the rejection in claim 1), Lehman discloses the method and apparatus further including specifying *c* *call-tasks*, *c* being less than or equal to *n*, the task scheduler scheduling a *call-task* when another task requests that the *call-task* be executed (e.g., Col. 7, Lines 29-32; Col. 16, Lines 21-23)

8. **As to claim 7** (Previously Presented) (incorporating the rejection in claim 1), Lehman discloses the method and the apparatus where tasks are given priority values such that whenever the task scheduler chooses between scheduling multiple tasks, all of which being ready to be executed, said task scheduler chooses from among those tasks that have the highest priority values (e.g., Col. 2, Lines 36-39; Col. 9, Lines 62-68; Col. 10, Lines 1-2; Col. 32, Lines 5-54)

9. **As to claim 15** (Currently Amended), Lehman discloses an apparatus for developing a real-time operating system comprising:

a computer (e.g., Fig. 27 – multi-process controller, Lines 30-32); a computer readable medium in data communication with the computer (e.g., Col. 135, Line 15 through Col. 137, Lines 64), the computer readable medium including a software synthesis program stored thereon (e.g., Col. 135, Line 15 through Col. 137, Lines 64), which when executed by the computer causes the computer:

Although Lehman discloses to specify a set of *n* tasks (e.g., Col. 1, Lines 33-38), task (1) through task (n), to be scheduled for execution; specify a scheduling algorithm (e.g., Col.3, Lines 36-39; Col. 9, Lines 56-61; Col. 32, Lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, Lines 29-32; Col. 16, Lines 21-23; Col. 2, Lines 36-39; Col. 9, Lines 62-68; Col. 10, Lines 1-2; Col. 32, Lines 5-54) for scheduling the execution of the set of *n* tasks; and synthesize source

code with to implement a task scheduler (e.g., Fig. 4, element 24; Fig. 24; Fig. 26; Col. 2, Lines 36-39; Col. 3, Lines 36-39; Col. 9, Lines 56-61; Abstract, Lines 20-25; Col. 10, Lines 8-12) that uses the scheduling algorithm and for controlling execution of the set of *n* tasks; but does not explicitly disclose other limitations stated below.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier discloses synthesizing source code from commands embedded in source code to implement the task scheduler (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values), and synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of "Operating System Library", " Code Selector", "Code Expander", "Targeted Operating Systems Code"; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker

and generates a makefile (for each processor) that includes the two code lists of OS and application SW)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into the Lehman's system to further provide other limitations stated above in the Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3)

Furthermore, Lehman and Gauthier do not explicitly disclose other limitations stated below.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses:

specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as an independent thread, with an init() method that initializes hardware resources and a start() method that registered itself to the run-time system …; 4th Para - Tasks … to init() and start() … stopExec() … A TASK data structure is also generated, which defines a scheduling entry representing a task …; Fig. 10 – Definition of a TASK in generated C code); and

the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., 'Scheduling Analysis' - … be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman- Gauthier's system to further provide other limitations stated above in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

Lastly, Liu discloses the challenges of developing embedded software for real-time control systems and argued that timing properties should be introduced at the programming level to bridge the gap between algorithm development and real-time priority tuning, but Lehman, Gauthier, and Liu do not further explicitly disclose other limitations stated below.

However, in an analogous art of *Design and Evaluation of an Embedded Real-time Micro-Kernel*, Singh discloses:

each task having an associated task context block (e.g., Sec. 2.2.7.1.1 The Model, 1st Para - … associate with each task a context; an identification string or number; a status; and a priority if applicable. These items are stored in a structure called a task-control block (or TCB) ...; Sec. 2.2.7.1.2 Task States, 1st Para – The operating system manages the task-control blocks by keeping track of the status or state of each task …); and

synthesizing source code from commands embedded in source code to manipulate task context blocks for said set of t init-tasks (e.g., Fig. 6.2 – Process data structure; Sec. 6.1 ECO Implementation, 3rd Para - … The information about the handles for incoming and outgoing data channels is obtained through the parameter p, which is a part of the Task Control Block (TCB) for the thread that executes the ECO …; P. 116 - This module allocates and initializes the data channels and Process control Blocks during the OS startup phase …; P. 119 – *allocate_ initialize_ process* – Function which allocates space to the Process Control Blocks and initializes them …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Singh into the Lehman-Gauthier-Liu's system to further provide other limitations stated above in the Lehman-Gauthier-Liu system.

The motivation is that it would further enhance the Lehman-Gauthier-Liu's system by taking, advancing and/or incorporating the Singh's system which offers significant advantages that it can take advantage of dual-register-set hardware to drastically reduce context-switching overhead; it also provides

support for dynamic priorities, "firing rules" for specifying the data channel conditions; and the reconfigurable options provided by DARK facilitate selective removal of unneeded features to improve performance, without requiring any change to the applications as once suggested by Singh (e.g., Sec. 1.2 Problem Statement, 3rd Para)

10. **As to claim 17** (Previously Presented) (incorporating the rejection in claim 15), Lehman discloses the apparatus being further configured to specify *f f-loop* tasks, each having an associated integer value *l(i)* for *i* ranging from *1* to *f* and *f* being less than or equal to *n* (e.g., Col. 20, line 63 through Col. 21, line 20 – for loops using an incrementing or decrementing counter, i.e. Loop for *I* = 1 to *X* (executing) block of statements), the task scheduler including a continuously executing loop such that each *f-loop* task executes exactly once every *l(i)* times that the loop is executed (e.g., Col. 21, Lines 13-19)

11. **As to claim 19** (Previously Presented) (incorporating the rejection in claim 15), please refer to above claim **5** accordingly.

12. **As to claim 21** (Previously Presented) (incorporating the rejection in claim 15), please refer to above claim **7** accordingly.

13. **As to claim 22** (Currently Amended), Lehman discloses an apparatus for developing a real-time operating system (e.g., Fig. 1; Col. 1, Lines 46-48; Col. 4, Lines 63-68; Col. 5, Lines 1-2) comprising:

a computer;

a computer readable medium in data communication with the computer, the computer readable medium including a software synthesis program stored thereon, the software synthesis program including:

means for specifying a set of n tasks (e.g., Col. 1, Lines 33-38), task (1) through task (n), to be scheduled for execution (e.g., Abstract, Lines 8-14; Col. 3, Lines 1-8; Col. 5, Lines 5-12; Col. 135, Lines 17-24); means for specifying a set of n tasks, means for specifying a scheduling algorithm for scheduling the execution of said the of *n* tasks (e.g., Col.3, Lines 36-39; Col. 9, Lines 56-61; Col. 32, Lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, Lines 29-32; Col. 16, Lines 21-23; Col. 2, Lines 36-39; Col. 9, Lines 62-68; Col. 10, Lines 1-2; Col. 32, Lines 5-54)

Although Lehman discloses a) means for specifying a set of n tasks, task (1) through task (n), to be scheduled for execution, at least one of the tasks of the set of n tasks being a preemptive or a non-preemptive task; c) means for synthesizing source code with to implement the task scheduler that uses said scheduling algorithm and said for controlling execution of the set of *n* tasks; Lehman does not explicitly disclose other limitations stated below.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier discloses:

means for synthesizing source code from commands embedded in source code to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values; Fig. 1 – an example of OS-based SW implementation of multiple tasks – "scheduling"; Sec. 2 – Related Work, 2nd Para – there are three approaches in SW implementation from multi-task descriptions; the first two approaches user OS as a scheduler and an interface of multiple tasks to the target architecture; bullets 1 through 3); and

said synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of "Operating System Library", " Code Selector", "Code Expander", "Targeted Operating Systems Code"; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and

assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into Lehman's system to further provide other limitations stated above in the Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3)

Furthermore, Lehman and Gauthier do not explicitly disclose other limitations stated below.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses:

specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as an independent thread, with an init() method that initializes hardware resources and a start() method that registered itself to the run-time system …; 4th Para - Tasks … to init() and start() … stopExec() … A TASK data structure is also generated, which defines a

scheduling entry representing a task ...; Fig. 10 – Definition of a TASK in generated C code); and

the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., P. 71, 'Scheduling Analysis' - ... be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - ... providing the interface and scheduling code, the actor code becomes single threaded and self-contained ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman- Gauthier's system to further provide other limitations stated above in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

Lastly, Liu discloses the challenges of developing embedded software for real-time control systems and argued that timing properties should be introduced at the programming level to bridge the gap between algorithm development and real-time priority tuning, but Lehman, Gauthier, and Liu do not further explicitly disclose other limitations stated below.

However, in an analogous art of *Design and Evaluation of an Embedded Real-time Micro-Kernel*, Singh discloses:

each task having an associated task context block (e.g., Sec. 2.2.7.1.1 The Model, 1st Para - … associate with each task a context; an identification string or number; a status; and a priority if applicable. These items are stored in a structure called a task-control block (or TCB) ...; Sec. 2.2.7.1.2 Task States, 1st Para – The operating system manages the task-control blocks by keeping track of the status or state of each task …); and

means for synthesizing source code from commands embedded in source code to manipulate task context blocks for said set of t init-tasks (e.g., Fig. 6.2 – Process data structure; Sec. 6.1 ECO Implementation, 3rd Para - … The information about the handles for incoming and outgoing data channels is obtained through the parameter p, which is a part of the Task Control Block (TCB) for the thread that executes the ECO …; P. 116 - This module allocates and initializes the data channels and Process control Blocks during the OS startup phase …; P. 119 – *allocate_ initialize_ process* – Function which allocates space to the Process Control Blocks and initializes them …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Singh into the Lehman-Gauthier-Liu's system to further provide other limitations stated above in the Lehman-Gauthier-Liu system.

The motivation is that it would further enhance the Lehman-Gauthier-Liu's system by taking, advancing and/or incorporating the Singh's system which

offers significant advantages that it can take advantage of dual-register-set hardware to drastically reduce context-switching overhead; it also provides support for dynamic priorities, "firing rules" for specifying the data channel conditions; and the reconfigurable options provided by DARK facilitate selective removal of unneeded features to improve performance, without requiring any change to the applications as once suggested by Singh (e.g., Sec. 1.2 Problem Statement, 3rd Para)

14. **As to claim 24** (Previously Presented) (incorporating the rejection in claim 22), please refer to above claim **3** accordingly.

15. **As to claim 26** (Previously Presented) (incorporating the rejection in claim 22), please refer to above claim **5** accordingly.

16. **As to claim 28** (Previously Presented) (incorporating the rejection in claim 22), please refer to above claim **7** accordingly.

17. **As to claim 29** (New), Lehman discloses a machine-readable medium embodying instructions which, when executed by a machine, cause the machine to:

specify a set of n tasks (e.g., Col. 1, Lines 33-38), task(l) through task(n), to be scheduled for execution (e.g., Abstract, Lines 8-14; Col. 3, Lines 1-8; Col. 5, Lines 5-12; Col. 135, Lines 17-24)

Further, Lehman discloses specify a scheduling algorithm for scheduling the execution of said set of n tasks (e.g., Col.3, Lines 36-39; Col. 9, Lines 56-61; Col. 32, Lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, Lines 29-32; Col. 16, Lines 21-23; Col. 2, Lines 36-39; Col. 9, Lines 62-68; Col. 10, Lines 1-2; Col. 32, Lines 5-54) for scheduling the execution of the set of *n* tasks.

Furthermore, Lehman discloses at least one of the task of the set of n tasks being selected as a preemptive or a non-preemptive task (e.g., Col. 9, Lines 52-56; Col. 10, Lines 3-12; Col. 35, Lines 1-5; Col. 136, Lines 40-50); synthesizing source code to implement a task scheduler (e.g., Fig. 4, element 24; Fig. 24; Fig. 26; Col. 2, Lines 36-39; Col. 3, Lines 36-39; Col. 9, Lines 56-61; Abstract, Lines 20-25; Col. 10, Lines 8-12) that uses the scheduling algorithm for controlling execution of said n tasks, Lehman does not explicitly disclose other limitations stated below

However, in an analogous art of 'Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software', Gauthier discloses:

synthesize source code (e.g., Sec. 4.2 – Synthesis of Application Specific OS and SW targeting) from commands embedded in source code (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an

example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values) to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks (e.g., Fig. 1 – an example of OS-based SW implementation of multiple tasks – "scheduling"; Sec. 2 – Related Work, 2nd Para – there are three approaches in SW implementation from multi-task descriptions; the first two approaches user OS as a scheduler and an interface of multiple tasks to the target architecture; bullets 1 through 3); and

said synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of "Operating System Library", " Code Selector", "Code Expander", "Targeted Operating Systems Code"; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into the Lehman's system to further provide other limitations stated above in the Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant

advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3).

Furthermore, Lehman and Gauthier do not explicitly disclose other limitations stated below.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software,* Liu discloses:

specify t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as an independent thread, with an init() method that initializes hardware resources and a start() method that registered itself to the run-time system …; 4th Para - Tasks … to init() and start() … stopExec() … A TASK data structure is also generated, which defines a scheduling entry representing a task …; Fig. 10 – Definition of a TASK in generated C code); the task scheduler further controlling one execution of each of said set of t init- tasks (e.g., P. 71, 'Scheduling Analysis' - … be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman- Gauthier's system to further provide other limitations stated above in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

Lastly, Liu discloses the challenges of developing embedded software for real-time control systems and argued that timing properties should be introduced at the programming level to bridge the gap between algorithm development and real-time priority tuning, but Lehman, Gauthier, and Liu do not further explicitly disclose other limitations stated below.

However, in an analogous art of *Design and Evaluation of an Embedded Real-time Micro-Kernel,* Singh discloses:

each task having an associated task context block (e.g., Sec. 2.2.7.1.1 The Model, 1st Para - ... associate with each task a context; an identification string or number; a status; and a priority if applicable. These items are stored in a structure called a <u>task-control block</u> (or <u>TCB</u>) ...; Sec. 2.2.7.1.2 Task States, 1st Para – The operating system manages the task-control blocks by keeping track of the status or state of each task ...); and

synthesizing source code from commands embedded in source code to manipulate task context blocks for said set of t init-tasks (e.g., Fig. 6.2 – Process data structure; Sec. 6.1 ECO Implementation, 3rd Para - ... The information about the handles for incoming and outgoing data channels is obtained through the

parameter p, which is a part of the Task Control Block (TCB) for the thread that executes the ECO ...; P. 116 - This module allocates and initializes the data channels and Process control Blocks during the OS startup phase ...; P. 119 – *allocate_ initialize_ process* – Function which allocates space to the Process Control Blocks and initializes them ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Singh into the Lehman-Gauthier-Liu's system to further provide other limitations stated above in the Lehman-Gauthier-Liu system.

The motivation is that it would further enhance the Lehman-Gauthier-Liu's system by taking, advancing and/or incorporating the Singh's system which offers significant advantages that it can take advantage of dual-register-set hardware to drastically reduce context-switching overhead; it also provides support for dynamic priorities, "firing rules" for specifying the data channel conditions; and the reconfigurable options provided by DARK facilitate selective removal of unneeded features to improve performance, without requiring any change to the applications as once suggested by Singh (e.g., Sec. 1.2 Problem Statement, 3rd Para)

18. **As to claim 30** (Previously Presented) (incorporating the rejection in claim 29), please refer to above claim **3** accordingly.

19. **As to claim 32** (Previously Presented) (incorporating the rejection in claim 29), please refer to above claim **5** accordingly.

20. **As to claim 34** (New) (incorporating the rejection in claim 29), please refer to above claim **7** accordingly.

21. Claims 4, 18, 25, and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman, in view of Gauthier, Liu, Singh and Xu et al. *On Satisfying Timing Constraints in Hard-Real-Time Systems'*, 1991, ACM' (hereinafter 'Xu')

22. **As to claim 4** (Previously Presented) (incorporating the rejection in claim 1), Lehman discloses the method and apparatus including means for specifying "loops" mechanism (e.g., Col. 20, line 63 through Col. 21 line 20)

But, Lehman, Gauthier Liu, and Singh do not specifically disclose *p-loop* task.

However, in an analogous art, Xu discloses means for specifying *p-loop* tasks, each having an associated integer value *ti* for *i* ranging from *1* to *p* and *p* being less than or equal to *n*, the number *ti* representing a number of regular time units (e.g., Sec. 2, 3rd paragraph, Lines 1-4), said task scheduler including a timer that schedules each *p-loop* task i to be executed approximately once every *ti* time units (e.g., Sec. 2, 3rd paragraph, Lines 1-4; Sec. 2, 7^{th} paragraph, on page 133 – A periodic process *p* can be described by a quadruple(*rp, cp, dp,*

prdp), where *prdp* is the period, *cp* is the worse case computation time required by process *p*, *dp* is the deadline, *rp* is the release time).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Xu into the Lehman-Gauthier-Liu-Singh in order to provide a timing constraints mechanism in the Lehman-Gauthier-Liu-Singh system.

The motivation is that (a) pre-run-time scheduling is essential if we want to guarantee that timing constraints will be satisfied in a complex hard-real-time system, (b) appropriate algorithms for solving mathematical scheduling problems that address those concerns can be used to automate pre-run-time scheduling, (c) if the task of computing schedules is completely automated, it would be very easy to modify the system and re-compute new schedules in case changes are required by applications as once suggested by Xu (e.g., Abstract, Lines 1-3; Sec. 6, 3rd Para., 6th Para.)

23. **As to claim 18** (Previously Presented) (incorporating the rejection in claim 15), please refer to above claim **4** accordingly.

24. **As to claim 25** (Previously Presented) (incorporating the rejection in claim 22), please refer to above claim **4** accordingly.

25. **As to claim 31** (New) (incorporating the rejection in claim 29), please refer to above claim **4** accordingly.

26. Claims 6, 20, 27, and 33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman in view of Gauthier, Liu, Singh, Xu, and David Lake (US 2004/0045003 A1) (hereinafter 'Lake')

27. **As to claim 6** (Previously Presented) (incorporating the rejection in claim 1), Lehman discloses the method and the apparatus including means for further specifying *r* preemptive-tasks (e.g., Col. 9, Lines 52-56), *r* being less than or equal to *n*, said task scheduler including a timer mechanism that counts a specified period of time at which time if a preemptive-task is currently executing (e.g., Col. 35, Lines 7-14) and continuing the execution of preemptive-task (e.g., Col. 9, line 64 through Col. 10, line 2)

But Lehman, Gauthier, Liu, Singh, and Xu do not specifically disclose other limitations stated below.

However, in an analogous art, Lake discloses the task's state is stored and execution is given to the task scheduler to schedule another task until a later time when the task scheduler restores the state of said preemptive-task. However, in an analogous art, Lake discloses the task's state is stored and execution is given to said task scheduler to schedule another task until a later time when the task scheduler restores the state of said preemptive-task (e.g., Fig. 1; [0031]; [0026], Lines 1-9; [0036], Lines 1-6)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Lake into the

Lehman-Gauthier-Liu-Singh-Xu system in order to further provide other limitations stated above in the Lehman-Gauthier-Liu-Singh-Xu system.

The motivation is to have its stack pointer set to a pre-calculated worst-case value guaranteed to leave sufficient space in the stack beneath the stack pointer for any preemptive tasks for task suspended/restored operations as once suggested by Lake (i.e., Abstract)

28. **As to claim 20** (Previously Presented) (incorporating the rejection in claim 15), please refer to above claim **6** accordingly.

29. **As to claim 27** (Previously Presented) (incorporating the rejection in claim 22), please refer to above claim **6** accordingly.

30. **As to claim 33** (New) (incorporating the rejection in claim 29), please refer to above claim **6** accordingly.

Conclusion

31. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax

phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192

/Michael J. Yigdall/
Primary Examiner, Art Unit 2192